Booting Linux from DiskOnChip HOWTO

Rohit Agarwal

<rohdimp_24@rediffmail.com>

Vishnu Swaminathan

<Vishnu.Swaminathan@siemens.com>

20060907

Revision History

Revision 1.0	2006−09−07	Revised by: MG
Last review for LDP publication		

This document discusses how to make the Flash Drives Linux bootable. We will describe how to boot from such a drive, instead of from the normal hard drive.

Table of Contents

Table of Contents

1. Introduction

1.1. Why this document?

DiskOnChip (DOC) is a flash drive that is manufactured by M−Systems. The use of flash drives is emerging as a substitute for Hard Disks in embedded devices. Embedded Linux is gaining popularity as the operating system of choice in the embedded systems community; as such, there is an increased demand for embedded systems that can boot into Linux from flash drives.

Much of the documentation currently available on the subject is either incorrect or incomplete; the presentation of the information which is provided by such documents is likely to confuse novice users.

1.2. NFTL vs. INFTL

Another fundamental problem is that most of the documents assume the DiskOnChip to be a NFTL (NAND Flash Translation Layer) device, and proceed to describe the booting process for NFTL devices. DiskOnChip architectures come in two variants, each of which requires different booting procedures: NFTL and INFTL (Inverse NFTL). Dan Brown, who has written a boot loader known as DOCBoot, explains the differences between these variants in a README document, which is included with the DOCBoot package: http://ftp.arm.linux.org.uk/pub/people/dwmw2/mtd/cvs/mtd/docboot/.

An INFTL device is organized as follows:

> IPL
>
> Media Header
>
> Partition 0 (BDK or BDTL)
>
> (Optional) Partition 1(BDK or BDTL)
>
> .. Up to at most Partition 3

Under Linux MTD partitions are created for each partition listed in the INFTL partition table. Thus up to 5 MTD devices are created.

By contrast the NFTL device is organized as follows:

> Firmware
>
> Media Header
>
> BDTL Data

Under Linux, normally two MTD devices will be created.

According to the above excerpt, the process used by the boot loader when fetching the kernel image for an INFTL device is different from the method used for NFTL devices, since both devices have different physical layouts. (repetitive)

Using a 2.4.x kernel for an INFTL DiskOnChip device is complicated by the lack of native support in pre−2.6.x kernels (although native NFTL support is present). Such functionality is only available by *patching* the kernel; an approach which is ill−advised.

Patching the kernel with external INFTL support is discouraged; the developers of the MTD driver, the open source driver available for DiskOnChip, are apprehensive of this approach as well. For more information on this matter, feel free to peruse the mailing list conversation on the subject at http://lists.infradead.org/pipermail/linux−mtd/2004−August/010165.html.

The drivers that provide native INFTL support in the 2.6.x kernels failed to identify the DiskonChip device used for this exercise, and the following message was reported by the system:

```
INFTL no longer supports the old DiskOnChip drivers loaded via docprobe.
Please use the new diskonchip driver under the NAND subsystem.
```

So then we decided to use the drivers provided by M−Systems (manufacturer of DiskOnChip). However, according to the documentation provided by the vendor on these drivers, they were designed for NFTL devices only. As such, we decided to write this HOWTO which will address the use of INFTL devices. We have taken special care to remove any ambiguity in the steps and also tried to give reasons for the need of a particular step so as to make things logically clear. We have explained things in such a way that a person with less experience on Linux can also follow the steps.

1.3. Practical goals

This document aims to act as a guide to:

- Use M−Systems DiskOnChip 2000 TSOP as an additional storage drive along with an IDE HDD running Linux on it.
- Install Linux on DiskOnChip 2000 TSOP and boot Linux from it.
- Install the Development Tool−Chain so as to compile and execute programs directly on DiskOnChip.

The method described here has been tested for DiskOnChip 2000 TSOP 256MB and DiskOnChip 2000 TSOP 384MB.

2. Reference configuration

We used the following hard− and software:

1. VIA Eden CPU 1GHz clock speed 256MB RAM
2. RTD Enhanced Phoenix − AwardBIOS CMOS Setup Utility (v6.00.04.1601)
3. Kernel 2.4.18 source code downloaded from www.kernel.org/pub/linux/kernel/v2.4
4. 256 MB M−Systems DiskOnChip 2000 TSOP (MD2202−D256)
5. M−Systems TrueFFS Linux driver version 5.1.4 from http://www.m−sys.com/site/en−US/Support/SoftwareDownload/Driver+Download.htm?driver=linux_binary.5_1_
6. LILO version 22.3.2 (distributed with driver)
7. DiskOnChip DOS utilities version 5.1.4 and BIOS driver version 5.1.4 from http://www.m−sys.com/site/en−US/Support/SoftwareDownload/TrueFFS5.x/BIOSDOSdriverandtools.htm
8. Dual bootable Hard Disk with Knoppix 3.9 and Windows XP using Grub 0.96 as the Boot Loader
9. GNU GCC−2.95.3
10. Latest root_fs_i386 image from http://www.uclibc.org/downloads/root_fs_i386.ext2.bz2 or http://www.uclibc.org/downloads/root_fs_i386.ext2.tar.gz

3. Assumptions

We have made some assumptions related to working directories and mounting points which we would like to mention before listing the entire procedure for putting Linux on DiskOnChip.

- We will perform all the compilation in `/usr/src` of the host machine so downloading of the necessary files must be done into that directory.
- All the commands listed are executed assuming `/usr/src` as the present working directory.
- We will mount the DiskOnChip partition on `/mnt/doc`.
- The names of the directories will be exactly the same as the files that have been downloaded so the document will give the actual path as were created on the host system.
- DiskOnChip and DOC have been used interchangeably to mean M−Systems DiskOnChip 2000 TSOP.
- The DOS utilities have been downloaded and saved in a Windows partition directory.

4. Using M−Systems DiskOnChip 2000 TSOP as an additional storage drive in Linux

The following are the steps performed for this purpose.

4.1. Step 1: Patch the Kernel

Download a fresh copy of Kernel 2.4.18 from http://www.kernel.org/pub/linux/kernel/v2.4.

The kernel that is downloaded from the site does not have support for the M−Systems driver so we need to add this functionality. This is done by adding a patch to the kernel.

The steps to conduct patching are as follows:

1. Untar the kernel source file and the M−systems TrueFFS Linux driver version 5.14. If the source code is in `.tar.gz` format, use

   tar −xvzf linux−2.4.18.tar.gz

   If the source code is in `.tar.bz2` format, use

   bunzip2 linux−2.4.18.tar.bz2

   After using **bunzip2**, you will get a file named `linux−2.4.18.tar`. Untar it using the command

   tar −xvf linux−2.4.18.tar

   Unarchiving the driver is done using the command

   tar −xvzf linux_binary.5_1_4.tgz

   This results in the creation of two directories: `linux` and `linux_binary.5_1_4`.
2. The TrueFFS Linux driver package contains three different folders:

   - `Documentation`: this contains a PDF document describing the various functions of TrueFFS.
   - `dformat_5_1_4_37`: this contains a utility **dformat**, which is used to update the firmware on the DiskOnChip (DOC) and to create low level partitions on the DOC.
   - `doc−linux−5_1_4_20`: this contains patches, `initrd` scripts and other utilities.
3. Now apply the patch to the kernel. We will use the `linux−2_4_7−patch` file that is present in `linux_binary.5_1_4/doc−linux−5_1_4_20/driver`. The following commands are used for this purpose:

   cd linux_binary.5_1_4/doc−linux−5_1_4_20/driver

   patch −p1 −d /usr/src/linux < linux−2_4_7−patch

   This will create a directory named `doc` in the `linux/drivers/block` directory.

4. The patch created the `doc` directory, but did not copy the source files of the M−Systems driver, which are necessary in order to build the driver, into this directory. So execute the following command:

   cp `linux_binary.5_1_4/doc-linux-5_1_4_20/driver/doc/* /usr/src/linux/drivers/block/doc`

Kernel version

The patch will fail for kernels other than 2.4.18 since the source files where the patch is to be applied may be somewhat different in different kernels. The patch has been provided specifically for kernel 2.4.18.

Before moving on to Step 2, do the following:

* Login as root.
* Make sure that **gcc** version is 2.95.3 else the build will fail. Use **gcc `--version`** to check this. If your **gcc** version is different compile gcc−2.95.3. Refer to http://xlife.zuarvra.net.columns/20020316 for this purpose.

4.2. Step 2: Compile the Kernel

Complete the following tasks for compiling the kernel:

1. **cd `linux`**
2. **make** ***`menuconfig`***

   Check for the following options:

   ♦ In the "Block devices menu", select:

      ◊ M−Systems driver as module i.e. (M)
      ◊ Loopback device support as built−in i.e. (*)
      ◊ RAM disk support as built−in i.e. (*)
      ◊ Initial RAM disk (initrd) support as built .in i.e. (*)

   ♦ In the "Processor type and features menu", select "Disable Symmetric Multiprocessor Support".
   ♦ In the "filesystem menu", select:

      ◊ Ext3 journaling file system support as built−in
      ◊ DOS FAT fs support as built−in[a]
      ◊ MSDOS fs support as built−in[b]
      ◊ VFAT (Windows−95) fs support as built−in[c]

   File System Menu

   a,b,c options should be activated if you want to mount your MS Windows partition, else they can be left out. It is, however, generally recommended to use them.

   An excellent resource on kernel compilation is the Kernel Rebuild Guide.

   The configuration file, `linux/.config` should essentially contain the following lines (only a part of the config file has been given):

```
#
# Loadable module support
#
CONFIG_MODULES=y
CONFIG_MODVERSIONS=y
CONFIG_KMOD=y

#
# Processor type and features
#

# CONFIG_SMP is not set


#
# Memory Technology Devices (MTD)
#
# CONFIG_MTD is not set

#
# Block devices
#
# CONFIG_BLK_DEV_FD is not set
# CONFIG_BLK_DEV_XD is not set
# CONFIG_PARIDE is not set
# CONFIG_BLK_CPQ_DA is not set
# CONFIG_BLK_CPQ_CISS_DA is not set
# CONFIG_BLK_DEV_DAC960 is not set
CONFIG_BLK_DEV_LOOP=y
# CONFIG_BLK_DEV_NBD is not set
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_SIZE=4096
CONFIG_BLK_DEV_INITRD=y
CONFIG_BLK_DEV_MSYS_DOC=m

#
# File systems
#
# CONFIG_QUOTA is not set
# CONFIG_AUTOFS_FS is not set
# CONFIG_AUTOFS4_FS is not set
CONFIG_EXT3_FS=y
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
# CONFIG_UMSDOS_FS is not set
CONFIG_VFAT_FS=y
# CONFIG_EFS_FS is not set
# CONFIG_JFFS_FS is not set
# CONFIG_JFFS2_FS is not set
# CONFIG_CRAMFS is not set
CONFIG_TMPFS=y
# CONFIG_RAMFS is not set
CONFIG_ISO9660_FS=y
# CONFIG_JOLIET is not set
# CONFIG_HPFS_FS is not set
CONFIG_PROC_FS=y
# CONFIG_DEVFS_FS is not set
# CONFIG_DEVFS_MOUNT is not set
# CONFIG_DEVFS_DEBUG is not set
CONFIG_DEVPTS_FS=y
# CONFIG_QNX4FS_FS is not set
# CONFIG_QNX4FS_RW is not set
```

```
# CONFIG_ROMFS_FS is not set
CONFIG_EXT2_FS=y
```

3. **make *`dep`***
4. **make *`bzImage`***
5. **make *`modules`***
6. **make *`modules_install`***
7. Copy the newly created `bzImage` to the `/bott` directory and name it `vmlinuz-2.4.18`, using this command:

   cp `/arch/i386/boot/bzImage /boot/vmlinuz-2.4.18`

Check for `lib/modules/2.4.18/kernel/drivers/block/doc.o`. This is the M−Systems driver that we need to access DiskOnChip.

4.3. Step 3: Create Nodes

Now we will create block devices, which are required to access the DOC These block devices will use the M−Systems driver that was built in Section 4.2 to access the DOC. The script **mknod_fl** in `linux_binary.5_1_4/doc-linux-5_1_4_20/driver` is used for this purpose.

We need to create the block devices with the major number of 62. For this purpose we will pass the argument 62 while creating the nodes:

./mknod_fl *`62`*

This will create the following devices in `/dev/msys` with major number 62:

```
fla...fla4
flb...flb4
flc...flc4
fld...fld4
```

4.4. Step 4: Reboot with the new kernel

In order to have the DiskOnChip recognized by Linux OS, we need to insert the DOC driver module into the kernel. Since the currently running kernel doesn.t have support for the M−Systems Driver, we need to boot into new kernel we just compiled in .

For this purpose we need to add the following entries in the `/boot/grub/menu.lst` file:

```
title Debian GNU/Linux,Kernel 2.4.18
root (hd0,7)
kernel /boot/vmlinuz-2.4.18 root=/dev/hda8
safedefault
boot
```

Where (hd0,7) is the partition holding the kernel image `vmlinuz-2.4.18` and `/dev/hda8` is the partition holding the *root filesystem*. These partitions may vary from one system to another. Now reboot and choose the kernel 2.4.18 option (the kernel that has been compiled in Step 2) in the **grub** menu to boot into the new kernel.

4.5. Step 5: Insert M−Systems Driver/Module in the new Kernel

The M−Systems driver by default gets loaded with major number 100, but our newly created nodes (see Section 4.3) have a major number 62. Therefore we need to insert this module with a major number 62. This can be done in either of two ways:

1. While inserting the module using **insmod** also mention the major number for the module which needs to be assigned to it otherwise it will take the default major number of 100:

   insmod ***`doc major=62`***
2. Add the following line to `/etc/modules.conf`:
   ```
   options doc major=62
   ```
   Then use **modprobe** ***`doc`*** to insert the modules.

Check for the correct loading of the module using the **lsmod** command without options.

4.6. Step 6: Create a filesystem on the DiskOnChip

Before we can start using DiskOnChip we need to create a filesystem on it. We will create an ext2 filesystem since it is small in size.

This involves a hidden step of making partitions on the DOC using **fdisk**. The actual steps are as follows:

1. **fdisk** **`/dev/msys/fla`**

   This command will ask to create partitions. Create a primary partition number 1 with start cylinder as 1 and final cylinder as 1002.

   Check the partition table, which should look like this:

   ```
   Device              Boot  Start       End         Blocks     ID      System
   /dev/msys/fla1             1          1002          255984    83      Linux
   ```
2. Make the filesystem on `/dev/msys/fla1` with the command

   mke2fs **`-c /dev/msys/fla1`**

   Where `fla1` is the first primary partition on the DOC. (We have created only one partition in order to avoid unnecessary complexity.)

4.7. Step 7: Mount the newly created partition to start accessing DOC

Create a new mount point for the DiskOnChip in the `/mnt` directory:

mkdir **`/mnt/doc`**

Mount the DOC partition on the newly created directory:

mount -t *auto* /dev/msys/fla1 /mnt/doc

You will now be able to read and write to the DOC as an additional storage drive.

When you reboot your system, make the DOC available by inserting the driver into the kernel (see Section 4.5) and mounting the device.

5. Install Linux and LILO on DiskOnChip

In this section we will learn how to install Linux operating system on an unformatted DOC and boot from it using LILO as the boot loader.

In order to get to this state, a procedure will be discussed. Some steps in this procedure resemble the steps discussed previously in this document. Even so, this should be considered a separate procedure, rather than a continuation of the steps in Section 4.

In general, to make a device to boot into Linux, it should have the following components:

- Kernel Image
- Root Filesystem
- Boot loader to load the kernel Image into memory

This section will basically try to fulfill the above three requirements.

The following steps should be followed for achieving the goal of this section.

5.1. Step 1: Copying the DOC firmware onto DiskOnChip

We will use the **dformat** utility from `linux_binary.5_1_4/dformat_5_1_4_37`.

M−Systems does not provide the firmware for using the DOC on Linux platforms. We address this problem by making a copy of the firmware shipped with the M−Systems dos utilities into this directory ("dos utilities" is the term used by the M−Systems people so we have also used this name). On our system we copied it by mounting the windows partition and extracting it from there:

mount −t *auto*/dev/hda5 /mnt/d

cp /mnt/d/dos\ utilities/doc514.exb linux_binary.5_1_4/dformat_5_1_4_37/

Now format the drive, using the **dtformat** from `linux_binary.5_1_4/dformat_5_1_4_37/`:

cd linux_binary.5_1_4/dformat_5_1_4_37/

./dformat −WIN:D000 −S:doc514.exb

D000 specifies the address of the DiskOnChip in the BIOS.

The following is the BIOS (RTD Enhanced Phoenix − AwardBIOS CMOS Setup Utility (v6.00.04.1601)) setting on our system.

The Integrated peripherals of the BIOS menu should have:

```
SSD Socket #1 to Bios Extension
Bios Ext. Window size 8k
Bios Ext. window [D000:0000]
Fail safe Boot ROM [Disabled]
```

The Bios Ext. Window denotes the address for your DiskOnChip.

BIOSes

The setting may be different depending upon your BIOS version.

Now shutdown the system and boot into Windows XP.

From now on you will notice the TrueFFS message and some time delay before the Grub Menu appears.

5.2. Step 2: Format DiskOnChip using Dos Utilities

Boot into Windows XP. We will use the M−Systems Dos Utilities for formatting the DiskOnChip. The Dos utility **dformat** will copy the firmware to the DOC, and then format it as a *fat16* device.

Using the command prompt, run the following command from the DOS utilities folder (assuming that you have already downloaded the DOS utilities):

dformat `/WIN:D000 /S:doc514.exb`

Check the DOC partition using another utility called **dinfo**. A sample **dinfo** output is given in the appendix.

Again shutdown the system and now boot into Linux.

Always shutdown

After formatting you should always do a full shutdown (power off) and not just a reboot.

Even though Step 1 and Step 2 seem to be the same, the only difference being that Step 1 is done from Linux and Step 2 from Windows XP, they both have to be done.

5.3. Step 3: Patch and Compile the kernel 2.4.18

This has to be performed in exactly the same manner as described in Section 4.1 and Section 4.2.

Also add an entry for the new kernel in `/boot/grub/menu.lst` as described in Section 4.4.

5.4. Step 4: Create nodes

This is done using the ame procedure as described in Section 4.3.

5.5. Step 5: Modify the /etc/module.conf file

The file `/etc/modules.conf` has to be modified, adding this line at the end of the file:

```
options doc major=62
```

This is required since our nodes use a major number of 62, while the doc driver module uses a major number of 100. When creating the `initrd` image, the driver will be loaded with major number value of 100 (instead of 62) if you do not edit the module configuration file. This will make it impossible for the nodes to use the driver. The reason for using the initrd image will be explained in the next step.

The **mkinitrd_doc** script from `linux_binary.5_1_4/doc-linux-5_1_4_20/driver` reads the

/etc/modules.conf file and looks for anything that has been mentioned for the DOC driver regarding the major number. By default, **mkinitrd_doc** will create an initrd image that loads the DOC module with a major number of 100. However, with the modifications we have made to the /etc/modules.conf file, the initrd image will load the module with a major number of 62.

5.6. Step 6: Create the initrd image

Run the **mkinitrd_doc** script from linux_binary.5_1_4/doc-linux-5_1_4_20/driver/:

./mkinitrd_doc

This may give warning messages similar to the following, which can be safely ignored:

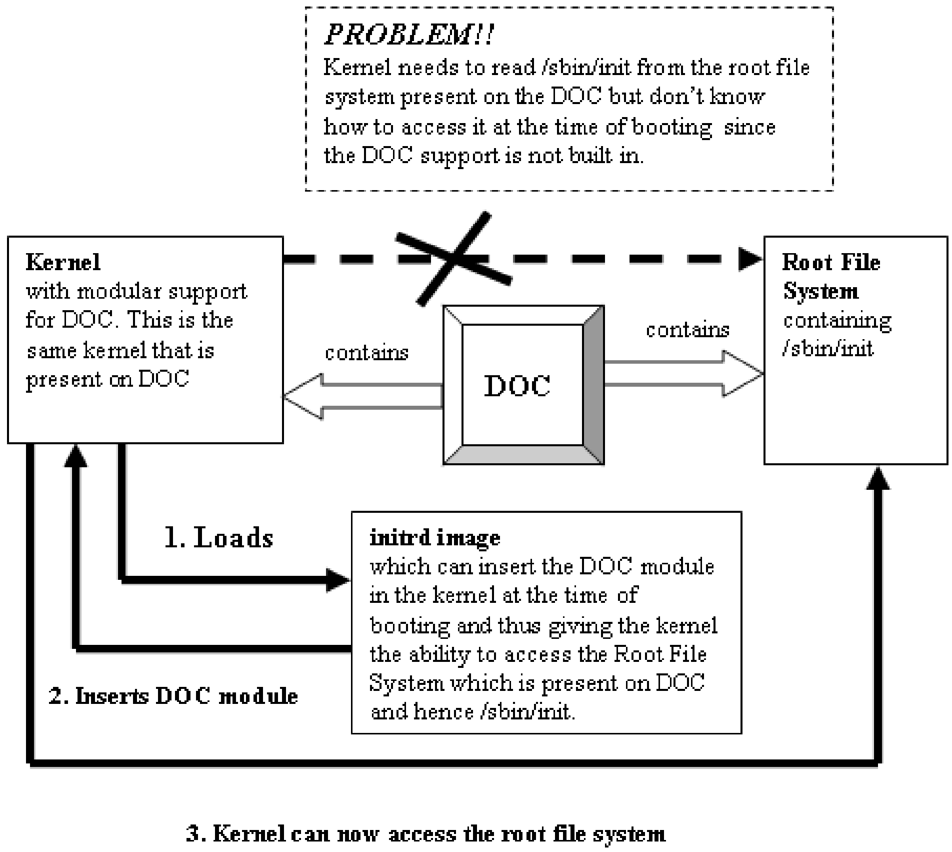
```
cp: cannot stat ./sbin/insmod.static.: No such file or directory
cp: cannot stat ./dev/systty.: No such file or directory
```

Check for the newly created initrd image, initrd-2.4.18.img, in the /boot directory.

Running the **mkinitrd_doc** script produces this image. The reason for making an initrd image is that the provided M−Systems driver cannot be added as a built−in support in the kernel, which leaves no other option than adding it as a loadable module. If we want to boot from DOC, the kernel should know how to access the DOC at the time of booting to search for **/sbin/init** in the root filesystem on the DOC (the root filesystem is necessary to get the Linux system up).

In the booting sequence of the Linux, **/sbin/init** is the file (a command actually) that the kernel looks for in order to start various services and, finally, give the login shell to the user. The figure below illustrates the problem:

Figure 1. Why we need an initrd image



5.7. Step 7: Insert the DOC driver into the new kernel

Reboot the system and boot into the newly created kernel.

Now insert the doc module:

modprobe *doc*

This will give the following messages:

```
fl : Flash disk driver for DiskOnChip
fl: DOC devices(s) found: 1
fl: _init:registed device at major 62
.
.
.
.
```

To access the DOC, ensure that the major number assigned to the nodes is 62.

In case of a major number of 100 is assigned, check if the `/etc/modules.conf` was successfully modified. If it was not, then repeat Section 5.5. You must then also repeat Section 5.6 because the `initrd` image depends on `/etc/modules.conf`. If the DOC entry were incorrect in this file, the `initrd` image will be useless.

5.8. Step 8: Create a filesystem on the DiskOnChip

Perform Section 4.6. This is required to create partitions on the DOC.

5.9. Step 9: Build Root Filesystem on the DiskOnChip

Before starting with this step make sure that you have not mounted `/dev/msys/fla1` on any of the mount points, as this step will involve reformatting the DiskOnChip.

Also, in order to understand the details of Root File system refer to The Linux Bootdisk How To available at http://www.tldp.org.

We will use the **mkdocimg** script located in `linux_binary.5_1_4/doc-linux-5_1_4_20/build`.

We will also use the `redhat-7.1.files` directory, located in the same directory (i.e. `build`), which contains the list of the files that will be copied in the root filesystem that will be created on the DOC.

./mkdocimg `redhat-7.1.files`

This step will take a few minutes to complete.

Now mount the `/dev/msys/fla1` partition on the mount point `/mnt/doc` and check the files that have been created:

mount `-t` *`auto`* `/dev/msys/fla1 /mnt/doc`

cd `/mnt/doc`

The following directories are created on the DOC as a result of running the script:

```
bin dev sbin etc lib usr home mnt tmp var boot
```

The most important is the boot directory. This contains the `vmlinuz-2.4.18` and `initrd-2.4.18.img` which gets copied from the `/boot` directory. This directory is required when booting from DiskOnChip.

Apart from these files there are some other files which must be deleted:

- `System.map-2.4.18`
- `boot.3E00`

These two files are created later by LILO.

The `redhat-7.1.files` directory contains a list of files and directories that will be created when we use the **mkdocimg** script.

This script does not create all the files that are necessary for creating the root filesystem on the DOC. So replace the directories created by the **mkdocimg** script, with the directories of the `/` filesystem (root filesystem that is currently running).

The directories under `/`, such as `etc`, `sbin`, `bin` and so on contain lot of files that are not useful and ideally should not be copied while building the root filesystem for DOC. But since we have not discussed the files that are essential and the files that can be removed, we therefore suggest that one should copy the entire contents of the directories. We know that it is a clumsy way of building the root filesystem and will unnecessarily take lot of memory; bear with us as in the next section we will explain how to put the development tools on the DOC. We will then remove the useless files from the root filesystem of DOC.

If you are aware of how to build the root filesystem we would encourage you to copy only the essential files.

The following is the set of commands we used to modify the root filesystem:

rm −rf /mnt/doc/sbin

rm −rf /mnt/doc/etc

rm −rf /mnt/doc/lib

rm −rf /mnt/doc/dev

cp −rf /sbin /mnt/doc

cp −rf /etc /mnt/doc

cp −rf /dev /mnt/doc

cp −rf /lib /mnt/doc

rm −rf /mnt/doc/lib/modules

Now our filesystem is ready.

The total size occupied by this filesystem will be about 35Mb.

5.10. Step 10: Use rdev to specify the DOC root filesystem location to kernel image

This step is required to specify the location of the DOC root filesystem to the kernel we compiled in the step 3. The step can be avoided by giving the details of the root filesystem location in the Boot Loader configuration file, but we had some problems in making the kernel locate the root filesystem at the time of booting so we recommend executing this command:

rdev /boot/vmlinuz−2.4.18 /dev/msys/fla1

5.11. Step 11: Compile lilo−22.3.2

We are going to use LILO as the boot loader since this is the only BootLoader that can read an INFTL device without many changes to be done to the BootLoader source code.

For more information on how LILO and other boot loaders operate, refer to .

We need to compile the lilo−22−3.2 source code to get the executable file for LILO.

We will use the source code from `linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2`.

Before starting the build we need to do the following:

1. Create a *soft link* for the kernel−2.4.18 source code with the name `linux`.

   When you untar the file `linux−2.4.18.tar.gz` it will create a directory `linux`. So we need to rename the directory `linux` to `linux−2.4.18` before creating a soft link with the same name:

   mv `linux linux−2.4.18`

   ln −s `linux−2.4.18 linux`

   If the above steps are not done the build might fail.
2. Patch file: `linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/common.h`:

   The lilo−22.3.2 source code that comes with the M−Systems `linux_binary.5_1_4.tgz` is buggy as one of the variables `PAGE_SIZE` is not defined. We need to patch the LILO source code as follows:

   Add the following lines in the `common.h` after the line "#include .lilo.h.":

   ```
   + #ifndef PAGE_SIZE
   + #define PAGE_SIZE 4096U
   + #endif
   #define 0_NACCESS 3
   ```
   Where "+" indicates the lines to be added.
3. Make sure that the **gcc** version is 2.95.3 by using **gcc `--version`**.

Now we can start the build process. Run

make *`clean`* && make

This will create a new LILO executable, `linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/lilo-22.3.2/lilo`. Copy this LILO executable into `/sbin/lilo` and `/mnt/doc/sbin/lilo`:

cp `linux_binary.5_1_4/doc−linux−5_1_4_20/lilo/lilo−22.3.2/lilo /sbin/lilo`

cp `linux_binary.5_1_4/doc−linux−5_1_4_20/lilo/lilo−22.3.2/lilo /mnt/doc/sbin/lilo`

5.12. Step 12: Copy the boot.b file into boot directory of DOC

We need to copy the file `boot.b` from `linux_binary.5_1_4/doc-linux-5_1_4_20/lilo/` to `/mnt/doc/boot`.

The file contains the essential *stage1* and *stage2* of the LILO boot loader.

5.13. Step 13: Modify the /etc/lilo.conf file

First, remove the existing `/etc/lilo.conf`:

rm -rf /etc/lilo.conf

Now create a new `/etc/lilo.conf`, using a text editor, and add the following lines to it:

```
boot=/dev/msys/fla
compact
install=/boot/boot.b
map=/boot/System.map
disk=/dev/msys/fla
bios=0x80
prompt
delay=50
timeout=50
image=/boot/vmlinuz
label=linux
root=/dev/msys/fla1
initrd=/boot/initrd.img
read-only
```

According to the above lines added to `/etc/lilo.conf`, one needs to create soft links for `vmlinuz-2.4.18` and `initrd-2.4.18.img` in `/mnt/doc/boot`:

cd /mnt/doc/boot

ln -s vmlinuz-2.4.18 vmlinuz

ln -s initrd-2.4.18.img initrd.img

Copy the newly created `/etc/lilo.conf` to `mnt/doc/etc/lilo.conf`.

5.14. Step 14: Store the new LILO configuration on the DiskOnChip

This step will configure LILO in the MBR of the DiskOnChip and hence make the DiskOnChip bootable.

Ensure that `/dev/msys/fla1` (root filesystem partition for the DOC) is mounted on the mount point `/mnt/doc`.

Execute the following command to store the LILO configuration on the DOC:

lilo-v -C /etc/lilo.conf -r /mnt/doc

`/mnt/doc` denotes the location where the BootLoader will be installed, so it is installed on the DiskOnChip, as `/mnt/doc` is the mount point for the primary partition of DOC where LILO was configured.

It will create the following two files in the boot directory of DOC (i.e. `/mnt/doc/boot`):

1. `System.map-2.4.18`
2. `boot.3E00`

Now you should make a backup of the entire DiskOnChip to allow for easy restore of the files damaged by possible fatal errors:

cd /home

tar -cvzf docimg.tgz /mnt/doc

This will create a compressed copy of all the contents of DiskOnChip with the name `docimg.tgz` in `/home`.

5.15. Step 15: Modify etc/fstab of DiskOnChip root file system

Open the `/mnt/doc/etc/fstab` file and edit the line where the mount point is `/`. Change that line to:

```
/dev/msys/fla1   /   ext2   rw.noauto 0 1
```

5.16. Step16: Update Firmware

This step is required to update the firmware of the DiskOnChip. We will use the **dformat** utility from `linux_binary.5_1_4/dformat_5_1_4_37`:

./dformat -W:D000 -S:doc514.exb -Y -NOFORMAT

Don't format!

The `NOFORMAT` flag is important otherwise it will reformat the DiskOnChip device, and the contents will be lost.

The above command will cause the DiskOnChip to boot in the absence of any other bootable device. So we need to remove the Hard Disk in order to allow the DOC to boot.

5.17. Step17: BOOT from DiskOnChip

Check your BIOS manual for enabling booting from a BIOS extension device i.e. DiskOnChip. On our system we had to disable the Hard Disk and CDROM and set the first bootable device as HDD−0.

Reboot the system after making the necessary changes in the BIOS.

You will get the LILO menu and on pressing enter Linux gets booted from DiskOnChip.

6. Install Development ToolChain on DiskOnChip

This section may be left out if the requirement is not to have a development environment on the DiskOnChip. The following steps will install the necessary libraries and tools that are required for developing and executing programs on DiskOnChip. This will completely eliminate the concept of having a host system and target system since now the complete application development can be done on the target system itself. For this purpose we will use uClibc which is a C library that has been developed primarily for embedded systems. Also since our root filesystem that was created in the previous section is bulky (35 MB) we will remove the unnecessary files and make it smaller, approximately 11MB.

We will use the Buildroot package from www.uclibc.org to replace the existing bulky root filesystem of DOC with a tiny filesystem and to install the necessary development ToolChain which includes uClibc library, gcc, g++, make, ncurses, busybox, GNU tar, GNU grep and the GNU coreutils . For more details on Buildroot refer to http://buildroot.uclinux.org/buildroot.html. The http://www.uclibc.org website provides a precompiled package containing all the tools, which can be downloaded and used. We will use the precompiled package, which is available as `root_fs` image. Refer to the uClibc FAQ for more details.

Follow these steps in order to get the software working on your DOS:

6.1. Step1: Obtain the latest copy of root_fs_i386.ext2

Download `root_fs_i386.ext2.tar.gz` from www.uclibc.org/downloads/root_fs_i386.ext2.tar.gz.

It is around 22MB in size. This actually gets decompressed to a 100MB size file.

Untar the file in `/usr/src`:

tar −xvzf root_fs_i386.ext2.tar.gz

This will create a file `root_fs_i386.ext2`.

We need to mount this file using a loopback device. Do the following steps:

mkdir root_fs

mount −o *loop* root_fs_i386.ext2 root_fs

Now you can access the content of the file `root_fs_i386.ext2` through the `root_fs` directory. The `root_fs` directory contains a number of directories which makes the root filesystem, like `bin`, `var`, `sbin`, `opt`, `root`, `home`, `etc`, `usr`, `lib`, `tmp`, `dev`, and `proc`.

The `usr` and `lib` directories contain the development tools like **gcc** and **g++**.

6.2. Step2: Replace the root filesystem of the DiskOnChip

Replace the `bin`, `var`, `sbin`, `etc`, `lib`, `usr`, `proc`, `mnt`, `home` and `opt` directories of the DiskOnChip with the ones of the `root_fs` image.

Do not replace boot and dev!

The `boot` directory of the DiskOnChip has to be kept intact since it contains the kernel image, `initrd` image and Map file that is used by LILO to load the kernel into memory.

The `dev` directory should also not be replaced since it contains the device nodes for DOC (The replacement task will take 5−10 minutes.)

6.3. Step3: Modify etc/fstab of DiskOnChip root file system

Open the newly replaced `/mnt/doc/etc/fstab` and edit the line where the mount point is `/`. Change that line to:

```
/dev/msys/fla1   /   ext2   rw.noauto 0 1
```

6.4. Step4: Reboot

Reboot from DiskOnChip and enjoy the uClibc development environment.

You will get the message

```
Welcome to the Erik.s uClibc development environment.
```

The entire root filesystem + boot directory + development tools take 84Mb of space.

7. References

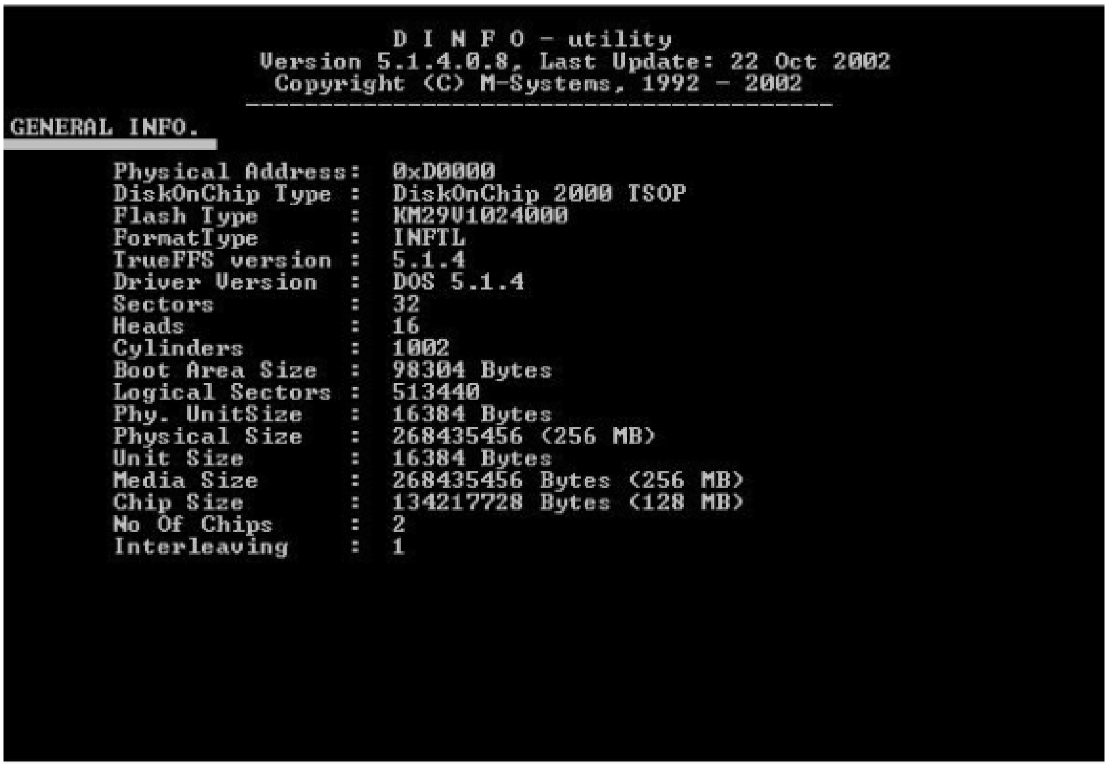
Apart from the web sites referenced, here are some books and documents we found to be useful:

- Application Note: Using DiskOnChip Under Linux With M−Systems Driver,RTD Embedded Technologies Inc.,SWM−640000016A
- Installation Manual for DiskOnChip TrueFFS driver for Linux ,M−Systems
- Building Embedded Linux Systems, Karim Yaghmour.O.reilly,First Edition, April 2003
- Installation Manual IM−DOC−021,Using the DiskOnChip with Linux OS,Ron Dick, Esther Spanjer, Vadim Khmelnitsky, M−Systems
- Installation Guide available with M−Systems TrueFFS Linux driver version 5.1.4
- Understanding the Linux Kernel, Daniel P. Bovet & Marco Cesati, O.reilly, Second Edition, March 2002.
- Booting Linux: The History and the Future, Werner Almesberger, June 25,2000

A. Output of dinfo

The following are the details of the internals of a Linux Bootable DiskOnChip 2000 TSOP 256MB (MD2202−D256) produced by the **dinfo** utility:

Figure A−1. Output of dinfo



According to the screenshot our DOC uses an INFTL translation layer.

B. License

C. About Authors

Rohit Agarwal has obtained his Masters in Information Technology from IIIT−Bangalore,India and Bachelors degree in Computer Science from I.E.T. Lucknow,India.He has co−authored another HowTo "Libdc1394 Library support for IEEE1394 camera HowTo" published by TLDP in January,2006. Visit his homepage for more details.

Visnu Swaminathan is a PhD and MS from Duke University,North Carolina,USA. He is currently working in Siemens Corporate Technology,India

authors can be contacted at

rohdimp_24@rediffmail.com (rohit)

vishnu.swaminathan@siemens.com

D. Dedications

I am grateful to all my friends for extending their support to carry out this work. I will specially thank Vikram and Chinmaya for their valuable advices.

Finally I will like to dedicate this document to Mr. S.Nagarajan my professor who inspired me to contribute to Open Source Community

Rohit Agarwal